Introduction to the Quantitative Analysis of Textual Data Using quanteda[*]

Kenneth Benoit and Paul Nulty

September 11, 2014

1 Introduction: The Rationale for quanteda

quantedais an R package designed to simplify the process of quantitative analysis of text from start to finish, making it possible to turn texts into a structured corpus, conver this corpus into a quantitative matrix of features extracted from the texts, and to perform a variety of quanttative analyses on this matrix. The object is inference about the data contained in the texts, whether this means describing characteristics of the texts, inferring quantities of interests about the texts of their authors, or determining the tone or topics contained in the texts. The emphasis of quantedais on *simplicity*: creating a corpus to manage texts and variables attached to these texts in a straightforward way, and providing powerful tools to extract features from this corpus that can be analyzed using quantitative techniques.

The tools for getting texts into a corpus object include:

- loading texts from directories of individual files
- loading texts "manually" by inserting them into a corpus using helper functions
- managing text encodings and conversions from source files into corpus texts
- attaching variables to each text that can be used for grouping, reorganizing a corpus, or simply recording additional information to supplement quantitative analyses with non-textual data
- recording meta-data about the sources and creation details for the corpus.

The tools for working with a corpus include:

- summarizing the corpus in terms of its language units
- reshaping the corpus into smaller units or more aggregated units
- adding to or extracting subsets of a corpus
- resampling texts of the corpus, for example for use in non-parametric bootstrapping of the texts (for an example, see Lowe and Benoit, 2013)

[*]This research was supported by the European Research Council grant ERC-2011-StG 283794-QUANTESS. Code contributors to the project include Alex Herzog, William Lowe, and Kohei Watanabe.

- Easy extraction and saving, as a new data frame or corpus, key words in context (KWIC)

For extracting features from a corpus, quantedaprovides the following tools:

- extraction of word types
- extraction of word *n*-grams
- extraction of dictionary entries from user-defined dictionaries
- feature selection through
  - stemming
  - random selection
  - document frequency
  - word frequency
  - and a variety of options for cleaning word types, such as capitalization and rules for handling punctuation.

For analyzing the resulting *document-feature* matrix created when features are abstracted from a corpus, quantedaprovides:

- scaling models, such as the Poisson scaling model or Wordscores
- nonparametric visualization, such as correspondence analysis
- topic models, such as LDA
- classifiers, such as Naive Bayes or *k*-nearest neighbour
- sentiment analysis, using dictionaries

quantedais hardly unique in providing facilities for working with text – the excellent tm package already provides many of the features we have described. quantedais designed to complement those packages, as well to simplify the implementation of the text-to-analysis workflow. quantedacorpus structures are simpler objects than in tm, as are the document-feature matrix objects from quanteda, compared to the sparse matrix implementation found in tm. However, there is no need to choose only one package, since we provide translator functions from one matrix or corpus object to the other in quanteda.

This vignette is designed to introduce you to quantedaas well as provide a tutorial overview of its features.

2 Installing quanteda

The code for the quantedapackage currently resides on `http://github/kbenoit/quanteda`. From an Internet-connected computer, you can install the package directly using the devtools package:

```r
library(devtools)
if (!require(quanteda)) install_github("quanteda", username = "kbenoit")
```

This will download the package from github and install it on your computer. For other branches, for instance if you wish to install the `dev` branch (containing work in progress) rather than the master, you should instead run

```
install_github("quanteda", username = "kbenoit", ref = "dev")
```

Typically, the `dev` branch of a software package is under active development — so while it contains the latest updates, it is more likely to have bugs. The `master` branch might be missing some of the newer features, but should be more reliable.

3 Creating a corpus

3.1 Loading Documents into Quanteda

From a directory of files

The quanteda package provides several functions for loading texts from disk into a quanteda corpus.

A very common source of files for creating a corpus will be a set of text files found on a local (or remote) directory. To load in a set of these files, we will load a corpus from a set of text files using information on attributes of the text that have been conveniently stored in the text document's filename (separated by underscores). For example, for our corpus of Irish budget speeches, the filename `2010_BUDGET_03_Joan_Burton_LAB.txt` tells us the year of the speech (2010), the type ("BUDGET"), a serial number (03), the first and last name of the speaker, and a party label ("LAB" for Labour).

To load this into a corpus object, we will use the `corpusFromFilenames` function, supplying a vector of attribute labels that correspond with the elements of the filename.

```
library(quanteda)
# tmpDir <- tempdir() # create a temporary directory for example files textfile
# <-
# 'https://github.com/kbenoit/quanteda/blob/dev/texts/irishbudgets2010.zip?raw=true'
# download.file(textfile, paste(tmpDir, 'irishbudgets2010.zip', sep='/'),
# method='curl', extra='-L') # download this zipped archive of texts unzip the
# file to the temporary folder unzip(paste(tmpDir, 'irishbudgets2010.zip',
# sep='/'), exdir=tmpDir) list the files unzipped list.files(paste(tmpDir,
# 'budget_2010', sep='/')) create a corpus from the files, parsing the filenames
# ieBudgets2010 <- corpusFromFilenames(paste(tmpDir, 'budget_2010', sep='/'),
# c('year', 'debate', 'number', 'firstname', 'lastname', 'party'), sep='_')

data(inaugCorpus)
# inaugCorpus <- subset(inaugCorpus, year=='2010')
```

This creates a new quanteda corpus object where each text has been associated values for its attribute types extracted from the filename:

```
summary(inaugCorpus)


## Corpus consisting of 57 documents, unindexed.


## Error:  arguments imply differing number of rows:  57, 0
```

From a vector of texts

Another method of creating a corpus from texts is to read texts into character vectors, and then create the corpus from these. The function get `getTextDir` takes a path to a directory containing some texts, and reads the texts into a character vector.

```
# load two vectors of texts (Bollinger texts from Evans et al JELS 2007) located
# in the texts directory of the github site amicusFile <-
# 'http://www.kenbenoit.net/courses/tcd2014qta/exercises/amicus_curiae.zip'
# tmpDir <- tempdir() download.file(amicusFile, paste(tmpDir,
# 'amicus_curiae.zip', sep='/'), method='curl', extra='-L') # download this
# zipped archive of texts unzip the file to the temporary folder
# unzip(paste(tmpDir, 'amicus_curiae.zip', sep='/'), exdir=tmpDir)

# load in the texts to a vector of texts using quanteda's getTextDir()
# amicusTexts <- c(getTextDir(paste(tmpDir, 'amicus/training', sep='/')),
# getTextDir(paste(tmpDir, 'amicus/testing', sep='/')))
```

In this case, the labels we are interested in (petitioner/respondent) are not clearly separated in the filename by underscores, but Now that we have the texts in a character vector, we can examine them and extract labels from the names. The code below uses the grep command, part of the standard R library, to make a list of labels from the names of the texts.

```
# change the encoding (because texts contain special symbols such as $)
# amicusTexts <- iconv(amicusTexts, from='latin1', to='UTF-8')

# examine the amicusTexts object - a named character vector where the names of
# the elements are the original text filename str(amicusTexts)

# set training class - Petitioner or Respondent, only known for the two test docs
# trainclass <- factor(c('P', 'R', rep(NA, length(amicusTexts)-2)))

# set test class, an attribute that could be used in classification here we take
# these from the text filenames, where 'AP' means Amicus brief for Petitioner
# 'AR' means Amicus brief for Respondent testclass <- rep(NA,
# length(amicusTexts)) # initialize the variable testclass[grep('AP',
# names(amicusTexts))] <- 'AP' testclass[grep('AR', names(amicusTexts))] <- 'AR'
```

Finally, we can create a corpus from the vector of texts, and the training and testing attributes.

```
# make a corpus object with texts and training and test labels amicusCorpus <-
# corpusCreate(amicusTexts, attribs = list(trainclass=trainclass,
# testclass=testclass), source = 'Bollinger texts from Evans et al JELS 2007',
# notes = 'Created as part of the quanteda vignette') summarize the first 10
# texts in the corpus summary(amicusCorpus, nmax=10)
```

There is also an interactive version of the getTextFiles() function, which will open a graphical file system browser to allow selection of the directory containing the files.

```
getTextDirGui()
```

We can also create a labelled corpus using the directory structure in which the files are stored. If the folder names in which the files are stored indicate values for a variable of interest.

todo corpus from folders*

3.2 Structure of a corpus in `quanteda`

A corpus contains attributes and metadata. Metadata is information associated with the entire set of texts, such as the source or date of creation. Metadata can also be used to package supplementary material with a corpus — for example, if the corpus analysis is part of a model that includes other forms of data, they can be included here.

The attributes of a corpus are the texts themselves, and any number of other attributes which may have different values for each text.

Adding new texts

We can add new texts to a corpus using the `corpusAppend` function. For example, if we wish to add another speech to the ieBudgets corpus, we can use a character vector for new text, and a data frame for the attributes of that text. The column names of the new attributes dataframe should match those in the existing corpus.

```
# newText <-'This is an example text to be added to the ieBudgets2010 corpus'
# newAttribs <- data.frame(year='2000', debate='bud', no='99', fname='Joe',
# speaker='Bloggs', party='Green' )

# ieBudgets2010 <- corpusAppend(ieBudgets2010, newText, newAttribs)
```

Adding new text attributes

Alternatively, we can add a new column of attribute values without appending any new texts.

```
newAttrib <- c("gray", "gray", "blue", "gray", "gray", "yellow", "gray", "gray",
    "gray", "gray", "gray", "gray", "gray", "gray", "gray")

ieBudgets2010 <- corpusAddAttributes(ieBudgets2010, newAttrib, name = "suitcolor")
names(ieBudgets2010$attribs)
```

4 Manipulating a corpus

5 Extracting Features

In order to perform statistical analysis such as document scaling, we must extract a matrix associating values for certain features with each document. In quanteda, we use the dfm function to produce such a matrix. [1].

By far the most common approach is to consider each word type to be a feature, and the number of occurrences of the word type in each document the values. This is easy to see with a concrete example, so lets use the `dfm` command on the full built-in Irish budget speeches corpus. In addition to indexing into the matrix with `:`, you can also view the matrix by clicking on the docMat variable in the RStudio Environment pane, or using the `View()` R command.

[1]dfm stands for document-feature matrix — we say 'feature' as opposed to 'term', since it is possible to use other properties of documents (e.g. ngrams or syntactic dependencies) for further analysis

```r
docMat <- dfm(ieBudgets2010)
docMat[1:5, 1:5]
```

6 Analyzing a document-feature matrix

By default, each row in the document-frequency matrix corresponds to a single document on disk. However, if our variable does not correspond with how the documents are stored as files, we can combine together texts and study them as if they were all part of the same document. For example, rather than counting word frequencies within each speech, we can count word frequencies as they are used by each party. We can do this using the `group` argument to `dfm`:

```r
partMat <- dfm(ieBudgets2010, group = "party")
partMat[1:3, 1:3]
```

We can now score and plot the parties using a statistical scaling technique, for example correspondence analysis (**?**).

```r
library(ca)
model <- ca(t(partMat), nd = 1)
dotchart(model$colcoord[order(model$colcoord[, 1]), 1], labels = model$colnames[order(model$colcoord[,
    1])])
```

We will discuss analysis techniques in more detail later. The dfm command has many optional arguments for applying standard pre-processing techniques to texts. We can choose to apply word stemming, which will sum together counts of words that have a common morphological root — for example *earn, earning, earns* and *earned* will all be counted together under the single stem *earn**. Again, this is easy to see by making a new matrix and inspecing it with `View()`

```r
docMatStems <- dfm(ieBudgets2010, stem = TRUE)
docMatStems[1:5, 1:5]
```

Importing to and exporting from tm

The tm package uses a sparse matrix format to store document-frequency matrices. Word frequency follows a power-law distribution (Zipf's law) — a few words are very frequent, while most words in the total corpus vocabulary don't occur at all in the a particular document, particularly if the length of each document is a small fraction of the length of the total corpus. The result is that most cells in a document-frequency matrix have a value of zero.

In order to use less storage space, tm only stores the row and column numbers where the value is not zero. This is known as a simple triplet representation. To make use of functions from tm, we must first convert the quanteda dfm into this format.

```r
tmMat <- dfm2tmformat(partMat)
names(tmMat)
```

References

Lowe, William and Kenneth Benoit. 2013. "Validating Estimates of Latent Traits From Textual Data Using Human Judgment as a Benchmark." *Political Analysis* 21(3):298–313.